Multi-CAST

case studies

Nils Norman Schiborr
University of Bamberg

Geoffrey Haig
University of Bamberg

July 2018
v1.0

LAC Language Archive Cologne

ARC CENTRE OF EXCELLENCE FOR THE DYNAMICS OF LANGUAGE

DFG

Multi-CAST

case studies

Nils Norman Schiborr
University of Bamberg

Geoffrey Haig
University of Bamberg

July 2018
v1.0

Multi-CAST Multilingual Corpus of Annotated Spoken Texts

Citation for this document

Schiborr, Nils N. & Haig, Geoffrey. 2018. Multi-CAST case studies. In Haig, Geoffrey & Schnell, Stefan (eds.), *Multi-CAST: Multilingual Corpus of Annotated Spoken Texts.* (archive handle) (date accessed)

Citation for the Multi-CAST collection

Haig, Geoffrey & Schnell, Stefan (eds.). 2018[2015]. *Multi-CAST: Multilingual Corpus of Annotated Spoken Texts.* (`https://lac.uni-koeln.de/en/multicast/`) (date accessed)

Archiving and versioning

This document has been archived at the *Language Archive Cologne* (LAC), accessible online at `lac.uni-koeln.de/en/multicast/`. The LAC is part of the Data Center for the Humanities (DCH) at the University of Cologne, Germany.

This is version 1.0 of the *Multi-CAST case studies*, last updated 1 July 2018. The latest version is always available from the LAC.

This document was typeset with XeLaTeX and v2.0-10 of the *multicast2* class.

Contents

1 Introduction

This document collects a number of case studies using the Multi-CAST collection, a database of annotated spoken language data from multiple languages, edited by Geoffrey Haig and Stefan Schnell (2015). Multi-CAST has been designed to address questions from areas of study such as discourse structure and referentiality. The Multi-CAST annotation data and documentation are hosted at the Language Archive Cologne (LAC),[1] a part of the Data Center for the Humanities at the University of Cologne and a project within the CLARIN-D framework.

The analyses showcased here draw from a relatively narrow field of inquiry; two of them in particular replicate analyses presented in a recent research paper (Haig & Schnell 2016b). Naturally, the Multi-CAST data can be leveraged for a wide variety of investigations beyond the few shown here. For a discussion of the theoretical motivations behind the Multi-CAST project, see the *Multi-CAST research context* (Haig & Schnell 2016a), and for a detailed description of the formal structure of the collection and the annotations therein, please refer to the *Multi-CAST structural overview* (Schiborr 2016).

This guide is structured as a step-by-step walkthrough that demonstrates some of the possible ways of working with Multi-CAST. Readers are advised to have familiarized themselves with the *GRAID manual* (Haig & Schnell 2014) and the *Multi-CAST structural overview* (Schiborr 2016) before continuing. The examples build on the R programming language and the *multicastR* companion package (Schiborr 2018),[2] but only a basic level of familiarity with statistical analysis and R is assumed. The methods discussed in the three case studies build on one another; it is thus best to start with the first one, then proceed to the others. As an aid, a list of basic GRAID symbols is provided in Appendix A, and Appendix B contains a short reference guide to regular expressions.

2 Case study: The lexicality of core arguments

2.1 Theoretical background

It has long been recognized that different syntactic roles are systematically associated with distinct pragmatic functions. The best-known such association is that of the subject role with topicality, or more generally with given, as opposed to new, information status (see Chafe 1976; 1980; 1994; Givón 1976; 1979; among many others).

1 Online at `https://lac.uni-koeln.de/en/multicast/`.

2 Gracious thanks go to Jenny Herzky, Nicholas Peterson, and Maria Vollmer for helping test and trial *multicastR* and the code in this guide. All remaining errors are our own.

This line of research was taken a step further in Du Bois (1987), where it is suggested that the subjects of transitive verbs (which we refer to as 'A') differ from the subjects of intransitive verbs ('S') in terms of information structure. While the former do exhibit the expected association with given information, the latter, according to Du Bois, are significantly less likely to be associated with given information. In fact, Du Bois suggests that the S role is actually specialized for accommodating new information. On this approach, then, it is not subjects in general that are associated with given information, but only transitive subjects (A). Du Bois (1987) suggests that the special role of transitive subjects in managing information flow in discourse actually mirrors the special role of transitive subjects in ergative alignment in grammar. Discourse, then, is supposedly organized along the lines of ergative alignment, involving a special role of A, in contrast to the apparent unity of S and direct objects ('P', also abbreviated as 'O' in some publications); see Du Bois (2003; 2017) for a detailed justification of the discourse basis of ergativity.

These proposals are tested in Haig & Schnell (2016b), combining a quantitative analysis of the Multi-CAST data with a meta-analysis of other published sources. The authors conclude that the assumption of a "discourse basis of ergativity" is in fact not justified. More specifically, while the data do confirm a particular information profile associated with the A role, we find little support for the claim that the S role is specialized for accomodating new information, or that it resembles the object role (P). The main results, considering only the Multi-CAST data, are illustrated in Figure 1. It will be seen that while A does indeed exhibit the predicted low figures for lexical expressions, there is no significant association of the P and the S role.

In Haig & Schnell (2016b), we also investigate the validity of different explanations for the findings, contrasting an approach based on lexical semantics (human vs. non-human) with an approach based on "information pressure", that is the tendency to minimize processing effort apparently associated with full noun phrases by distributing them across distinct clauses. The findings suggest that the semantic explanation provides a simpler explanation for the observed tendencies, rendering assumptions based on information pressure largely redundant; see Section 4 below for partial documentation of this aspect.

This study, largely based on the systematic cross-linguistic analysis of the parallel-annotated corpora in Multi-CAST, yielded a very significant challenge to an assumption that has been widely accepted for 30 years, and has prompted a re-assessment of the nature of the connection of grammar and discourse. In the following sections, we outline the main conceptual and methodological steps that were necessary to conduct this research and create the visualizations; please consult Haig & Schnell (2016b) and c for the details of argumentation and the relevant literature.

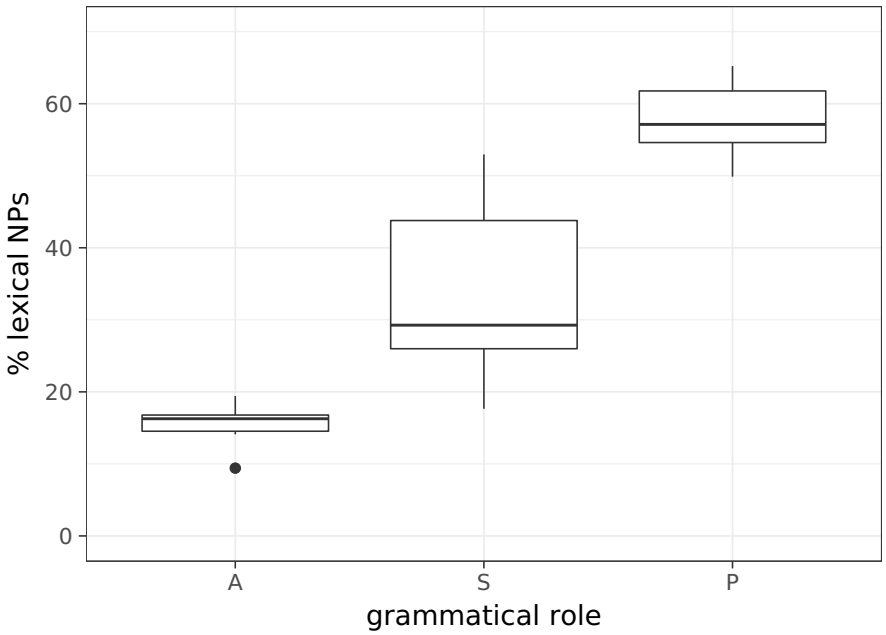

Figure 1 Percentage of lexical third-person expressions in A, S, and P role in the Multi-CAST collection.

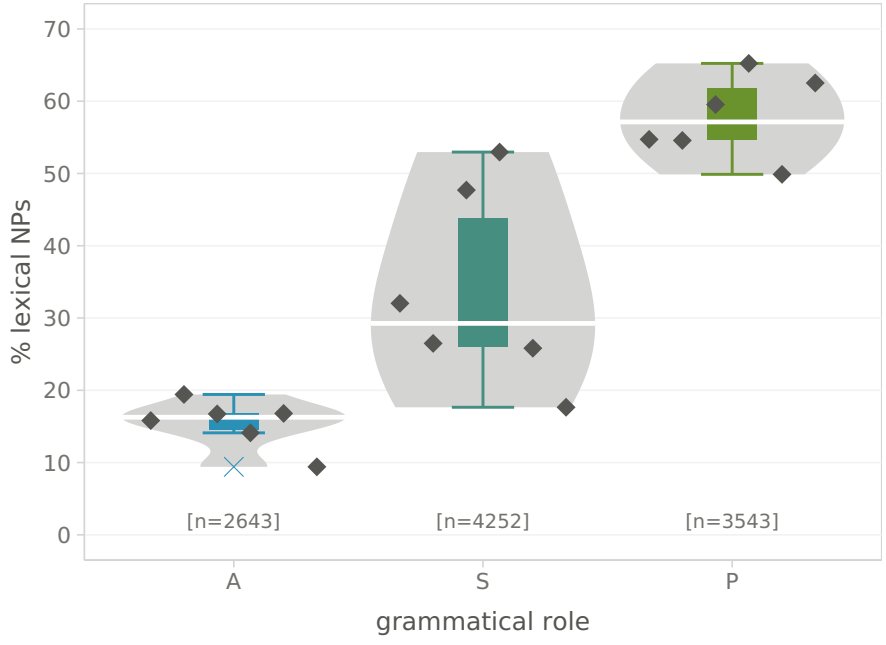

Figure 2 A more elaborate representation of Figure 1. Each dot represents a language in the sample.

2.2 Identifying S, A, and P

The main variables relevant for this research are the realizations of S, A, and P in the various corpora. We have followed much of the relevant research in assuming that a full noun phrase (including proper names) is approximately indicative of new information (*a woman, Mary Smith,* etc.) while pronouns (*she, they,* etc.), or zero anaphora, reflect given information. We refer to full noun phrases (those with a lexical head) as lexical expressions. The research question can thus be formulated as: what is the respective proportion of lexical expressions in each of the three main syntactic roles S, A, and P across different corpora? Do S and P cluster together, in opposition to A, as predicted by Du Bois, or do we find different configurations?

As for the methodology, the functions S, A, and P of all referring expressions (including zeroes) are systematically indicated in the GRAID annotations, so that the relevant figures can be automatically generated for each of the Multi-CAST corpora, using the steps indicated below. Nevertheless, certain analytical decisions need to be taken before the quantitative analysis can be conducted. For example, these include decisions regarding the status of various kinds of non-canonical subjects, which may be counted either as S or A, or excluded entirely, and regarding clausal constituents, which may exhibit object function, but which we have excluded from the counts; see Haig & Schnell (2016c). Similarly, we excluded first and second person arguments on the grounds that with these arguments, no contrast between a lexical and a non-lexical expression is available (they are exclusively pronominal, or zero). Once these general decisions have been made – and documented – the analysis can be conducted using the Multi-CAST data, which is most easily accessed with the help of the *multicastR* package. The necessary steps are outlined in the next sections.

2.3 Frequency calculations with R and *multicastR*

Before we can start working with the Multi-CAST data in R, we need to install the *multicastR* package.

```
install.packages("multicastR")
```

For help with the installation process, please refer to the relevant section of the *Multi-CAST structural overview* (Schiborr 2016).

Next we attach the *multicastR* package, then use it to retrieve the Multi-CAST annotation data from the internet. The first part can be achieved simply by declaring:

```
library(multicastR)
```

The second step involves the `multicast()` function provided by *multicastR*. This function downloads the corpus data from the servers of the Language Archive Cologne, and as such requires an active internet connection. It takes a single argument specifying which version of Multi-CAST to retrieve. For this demonstration we use the latest version of the data, at the time of writing the June 2016 (`1606`) release:

```
mc <- multicast("1606")
```

If no argument is given, the function selects the latest version by default; as such, the above is currently equivalent to simply calling `multicast()`. You can check which versions are available by invoking the `mcindex()` function.

As the annotation data files are multiple megabytes in size, it is advisable to store the output of `multicast()` in an object in the workspace as we have done here with `mc`, rather than call the function multiple times and download the data over and over again.

The output of the *multicastR* package's `multicast()` function is a `data.table`, a data structure provided by the package of the same name (Dowle & Srinivasan 2017). A `data.table` is similar to base R's `data.frame` and in fact inherits from it, but is significantly faster and more powerful. As we shall see, the *data.table* package provides numerous convenient facilities for handling large amounts of tabular data.

The first calculation we need to make is for the total number of expressions, lexical or otherwise, that match the criteria we outlined in Section 2.2 above, split by corpus. The use of `data.table`s makes this a simple matter thanks to their `dt[i, j, by]` syntax, which groups data conditionally by column. In `dt[i, j, by]`, `dt` is a `data.table` such as our `mc`; `i` are the conditions by which we select a subset of rows from the `dt` table; `j` is a column selector that allows operations to be performed on the selection; and `by` specifies by which columns the values of `j` should be grouped.

```
all <- mc[grepl("^[asp]$", gfunc) &
          grepl("^h$|^$", ganim) &
          !grepl("#", gform),          # i
          .N,                          # j
          by = c("corpus", "gfunc")]   # by
```

The resulting `data.table` `all` looks like this:

```
print(all)

      corpus gfunc    N
 1: cypgreek     s  231
 2: cypgreek     a  234
 3: cypgreek     p  466
 4:  english     s  555
 5:  english     a  412
 6:  english     p  962
 7:    nkurd     s  392
 8:    nkurd     p  388
 9:    nkurd     a  275
10:  persian     a  567
11:  persian     p  509
12:  persian     s  625
13:     teop     s  608
14:     teop     p  431
15:     teop     a  369
16:  tondano     s  266
17:    veraa     s 1841
18:    veraa     a  786
19:    veraa     p  787
```

The `mc` table produced by `multicast()` includes three extra columns that contain the segmented form (`gform`), person–animacy (`ganim`), and function symbols (`gfunc`) of the GRAID annotations. In table `all`, the data are grouped `by` the `corpus` and `gfunc` (GRAID function) columns. The `j` in our selection is `.N`, a *data.table* shorthand for the number of rows in each group of the selection, which is written to the column labelled `N` in the table.

Our `i` is made up of three conditions, all of which need to evaluate to `TRUE` for a row to be selected and subsequently counted. All three match regular expressions with the values of a particular column of the `mc` table, and employ the `grepl(pattern, data)` function to do so. A short reference guide for regular expressions is provided in Appendix B. The individual conditions are linked with the boolean `&` 'and' operator. The first selects those rows of the table for which the `gfunc` column contains only the GRAID functions ⟨`a`⟩, ⟨`s`⟩, or ⟨`p`⟩, and nothing else: the special characters `^` and `$` evaluate respectively to the beginning and end of a table cell, and nothing may come before or after them. The second condition does similarly for the `ganim` column, selecting ⟨`h`⟩ (third person human) `|` 'or' empty (third person non-human) glosses. The third and final condition is negated with `!`, and as such selects all but those rows that have a clause boundary marker ⟨`#`⟩ somewhere in the `gform` column. Note that the values in the `ganim` and `gfunc` columns do not contain the delimiters for GRAID person–animacy ⟨`.`⟩ and function symbols ⟨`:`⟩, so we do not need to match them.

Next we repeat the process, but add a condition to `i` that exclusively selects lexical expressions, this being those containing the ⟨np⟩ GRAID form symbol somewhere in the `gform` column. Because of this condition, we no longer need to explicitly exclude clause boundary markers ⟨#⟩ from the selection, and can hence remove the associated regular expression.

```
lex <- mc[grepl("^[asp]$", gfunc) &
          grepl("^h$|^$", ganim) &
          grepl("np", gform),          # i
          .N,                          # j
          by = c("corpus", "gfunc")]   # by
```

The table `lex` has the same format as `all`, but instead of counting all core arguments, it lists in its `N` column only the frequencies of lexical expressions.

We now have two tables containing the information we are interested in; the next step is to combine them into a single table with the `merge(x, y, by)` function. We specify `corpus` and `gfunc` as the columns to be merged `by`, as they are shared between the tables.

```
core <- merge(lex, all, by=c("corpus", "gfunc"), all = TRUE)
```

The resulting table `core` looks as follows (shortened):

```
print(core)

      corpus gfunc N.x  N.y
 1: cypgreek     a  37  234
 2: cypgreek     p 255  466
 3: cypgreek     s  74  231
---
17:    veraa     a  74  786
18:    veraa     p 492  787
19:    veraa     s 325 1841
```

Rather than leave the two frequency columns with the somewhat cryptic labels they currently have, it is a good idea to give them more descriptive names. Since we are working with a `data.table`, this is easily achieved by using the `setnames(data, old, new)` function:

```
setnames(core, c("N.x", "N.y"), c("nLex", "nAll"))
```

We calculate the percentage of lexical expressions by dividing the number of lexical NPs by the total number of all NPs in each role and corpus, then round the result:

```
core[, pLex := round(100 * nLex / nAll, 2)]
```

If we look closely at the `core` table, we can see that the Tondano corpus seems to only have values for S, and none for the A and P roles. The cause of this is that all instances of the ⟨:a⟩ and ⟨:p⟩ GRAID functions in the Tondano corpus are in fact subspecified with a suffix, and we have only been selecting rows that match the basic symbols exactly. Tondano is a language with a Philippine-type voice system, and hence has more than one "prototypical" transitive clause pattern, each receiving dedicated symbols. Please refer the Tondano annotation notes for details. In a proper investigation into the lexicality of core arguments and discourse ergativity, we might want to work around this, as we do below in Section 3, but for the sake of keeping this first demonstration as simple as possible, we will instead exclude Tondano from the analysis:

```
core <- core[corpus != "tondano", ]
```

The function `setorder(data, ...)` is another handy *data.table* tool. It allows us to sort data by any number of columns:

```
setorder(core, gfunc, corpus)
```

Here we sort alphabetically, first by GRAID function `gfunc`, then by `corpus` as tiebreaker. Let's take a look at the final output:

```
print(core)
```

```
      corpus gfunc nLex nAll  pLex
 1: cypgreek     a   37  234 15.81
 2:  english     a   80  412 19.42
 3:    nkurd     a   46  275 16.73
 4:  persian     a   80  567 14.11
 5:     teop     a   62  369 16.80
 6:    veraa     a   74  786  9.41
 7: cypgreek     p  255  466 54.72
 8:  english     p  525  962 54.57
 9:    nkurd     p  231  388 59.54
10:  persian     p  332  509 65.23
11:     teop     p  215  431 49.88
12:    veraa     p  492  787 62.52
13: cypgreek     s   74  231 32.03
14:  english     s  147  555 26.49
15:    nkurd     s  187  392 47.70
```

	A			S			P		
corpus	n(lex)	n(all)	p(lex)	n(lex)	n(all)	p(lex)	n(lex)	n(all)	p(lex)
C. Greek	37	234	15.8	74	231	32.0	255	466	54.7
English	80	412	19.4	147	555	26.5	525	962	54.6
N. Kurdish	46	275	16.7	187	392	47.7	231	388	59.5
Persian	80	567	14.1	331	625	53.0	332	509	65.2
Teop	62	369	16.8	157	608	25.8	215	431	49.9
Vera'a	74	786	9.4	325	1 841	17.7	492	787	62.5

Table 1 Lexicality of third-person expressions in A (subject transitive), S (subject intransitive), and P (direct object) role in six of the Multi-CAST corpora.

```
16:  persian     s  331  625 52.96
17:     teop     s  157  608 25.82
18:    veraa     s  325 1841 17.65
```

The table `core` as seen above is currently in what is called its long form, with one observation `pLex` per row. Although this long form is generally easier to work with in R, for the purpose of presenting the data it is usually better to reshape the table into its wide form, where related observations are given their own column. We can do so with the help of the `dcast()` function from the *data.table* package:

```
dcast(core,
      corpus ~ gfunc,
      value.var=c("nLex", "nAll", "pLex"))

# (output is too wide to be reproduced here)
```

The resulting wide form table is given in Table 1 with reordered columns. In the function `dcast(data, formula, value.var)`, `value.var` states explicitly which of the columns in the long form table contain values rather than category labels, in this case the two frequency counts `nLex` and `nAll` and their ratio `pLex`. The `formula` is an expression of the form [category column(s) by which values should be distributed across rows] `~` (tilde) [category column(s) by which values should be distributed across columns], which here is `corpus` (vertical) by `gfunc` (horizontal).

2.4 Visualization

Figures 1 and 2 show one suitable way of representing data of this kind, namely with a box-and-whisker plot. On the x-axis, it shows the three core grammatical roles A, S, and P, and on the y-axis the distribution across corpora of the percentage of lexical expressions in each role. With the following steps, we generate the simpler plot in Figure 1.

We will be using the *ggplot2* package for generating our plots (Wickham 2009). If you have not done so already, you will need to install the *ggplot2* package via `install.packages("ggplot2")`. As always, the first step is to load the package:

```
library(ggplot2)
```

Before we start plotting, we factorize the `gfunc` column. Factor levels can be assigned an inherent order by specifying `ordered = TRUE` and supplying an exhaustive ordered list of factor `levels`, here `"a"`, `"s"`, and `"p"`. The inherent hierarchization of a factorized vector or column is automatically respected when drawing graphs with by `ggplot()`. This way, we can specify the desired order of A, S, and P in the plot, rather than just having them ordered alphabetically.

```
core[, gfunc := factor(gfunc,
                       ordered = TRUE,
                       levels = c("a", "s", "p"))]
```

The following will generate three simple boxplots from the `core` table, with the three grammatical roles `a`, `s`, and `p` in the `gfunc` column on the x-axis, and the corresponding values in `pLex` on the y-axis. The first line specifies the data we indend to draw, the second instructs `ggplot()` to render it as a box-and-whisker plot, and the third (non-essential) line applies a simple style template.

```
lex.plot <- ggplot(data = core, aes(x = gfunc, y = pLex)) +
            geom_boxplot() +
            theme_bw()
```

To the definition of `lex.plot` we then add more suitable vertical scale limits for the y-axis and custom names and labels for both axes.

```
lex.plot <- lex.plot +
            scale_y_continuous(limits = c(0, 70),
                               name = "% lexical NPs") +
```

```
                scale_x_discrete(name = "grammatical role",                87
                                 labels = c("A", "S", "P"))                88
```

Calling the name of the plot object will render it; the output is reproduced above in Figure 1.

```
lex.plot                                                                   89
```

The plot in Figure 1 is as simple as it can be, which is as complex as it needs to be. Figure 2 above is an example of a somewhat more involved representation that adds dots for each data point, density data in the form of violin plots, and a splash of colour.

3 Case study: Null subjects

3.1 Theoretical background

While it is widely assumed that for most – if not all – languages, the category of 'subject' can be identified in syntax, languages differ quite radically in the conditions under which a subject requires overt expression. In the literature, two approaches to these differences can be discerned: a parametric approach and a discourse, or usage-based, approach.

The parametric approach goes back to Perlmutter (1971), who defined a pro-drop parameter, according to which a language either requires or does not require overt expression of referential subjects. The original either/or pro-drop parameter has since given way to more refined typologies, involving four distinct types of referential null subjects (RNS, cf. Holmberg 2009). It has also been extended to include referential null objects under the labels "radical pro-drop" or "discourse pro-drop" (e.g. Neeleman & Szendröi 2008).

In contrast to the parametric tradition, the second line of research is usage or discourse-based. On this view, RNS is a locus of gradual variation, thus not entirely determined by 'the grammar' of a language, but also dependent on contextual and interactional factors. Within language typology, this kind of research is associated with the work of Bickel and associates on referental density (RD, Bickel 2003; Stoll & Bickel 2009). RD is an empirical measure of the overall density of overt argument expressions in actual discourse, and is not restricted to subjects and objects. Within variationist sociolinguistics, referential null subjects, generally measured in the complementary value of rates of overt subject expression, have been extensively investigated as a linguistic variable, most notably across different varieties of Spanish; see Pešková (2013: 120–121) and Carvalho et al. (2015) for discussions of the relevant literature.

The Multi-CAST data readily lend themselves to a quantitative cross-linguistic approach to referential null subjects, and for those corpora which con-

sist of a larger number of texts by different speakers, we can also gain some insight into the range of language-internal variation in rates of zero subject realization. Figure 3 provides a visualization of overall rates of zero subject realization in seven corpora from the Multi-CAST. The values for individual texts are overlaid as dots, and give an idea of the levels of data reliability. It is evident – and hardly surprising – that individual languages do indeed differ, but we cannot readily identify particular types; rather we find gradient levels of zero subjects, both within and across languages. We also note, again unsurprisingly, that English in particular has a low rate of zero subjects, but nevertheless not as low as is often assumed. In conducting this research, it turns out that much depends on the coding and analytical decisions concerning what is to count as a zero subject cross-linguistically. We discuss these issues, and the actual steps required to extract and visualize these data from Multi-CAST, in the following sections.

3.2 Identifying null subjects

In Multi-CAST, the concept of 'subject' equates to all arguments which have been annotated with the function glosses ⟨:s⟩ or ⟨:a⟩. A null subject, as opposed to a non-null subject, is one that contains the form gloss ⟨0⟩. We assume the existence of a null subject if the following three conditions are met:

1. the predicate must license the argument in question;
2. the intended referent must be specific and retrievable from the discourse context; and
3. the predicate–argument construction must not systematically suppress the argument function in question.

3.3 Frequency calculations

Like in Section 2.3, the first step is to load the *multicastR* package into R, and then to retrieve the annotation data from the servers of the Language Archive Cologne. As before, be specify `"1606"` as the version key.

```
library(multicastR)

mc <- multicast("1606")
```

This step can of course be skipped if the package and data are already on hand.

In order to be able to analyze the realization of subjects, we first need to be able to identify them. The regular expression in the following line of code allows us to match all rows of the table `mc` that have the ⟨:a⟩ or ⟨:s⟩ role or

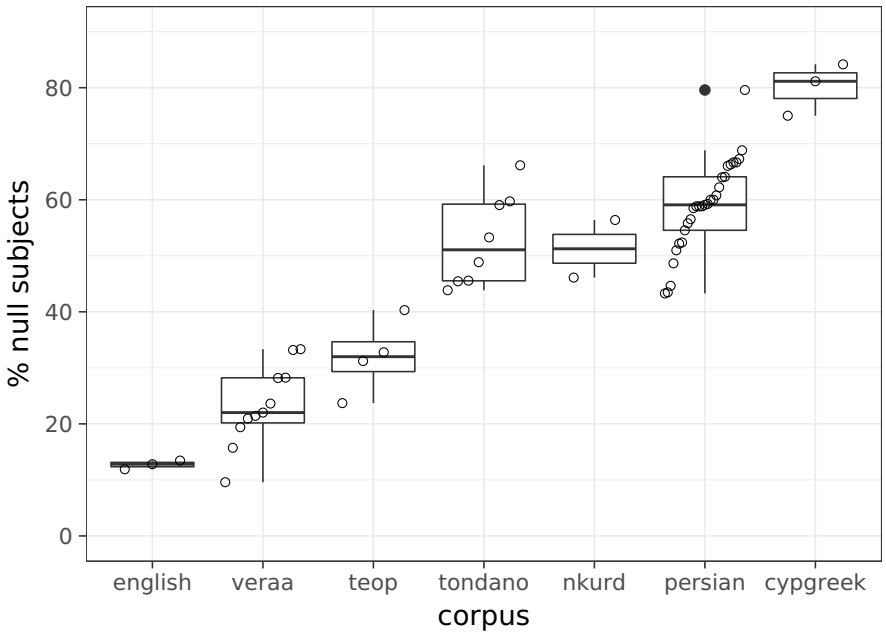

Figure 3 Distribution of null subjects (A+S) by corpus text in the seven Multi-CAST corpora. Each dot represents a single text in the collection.

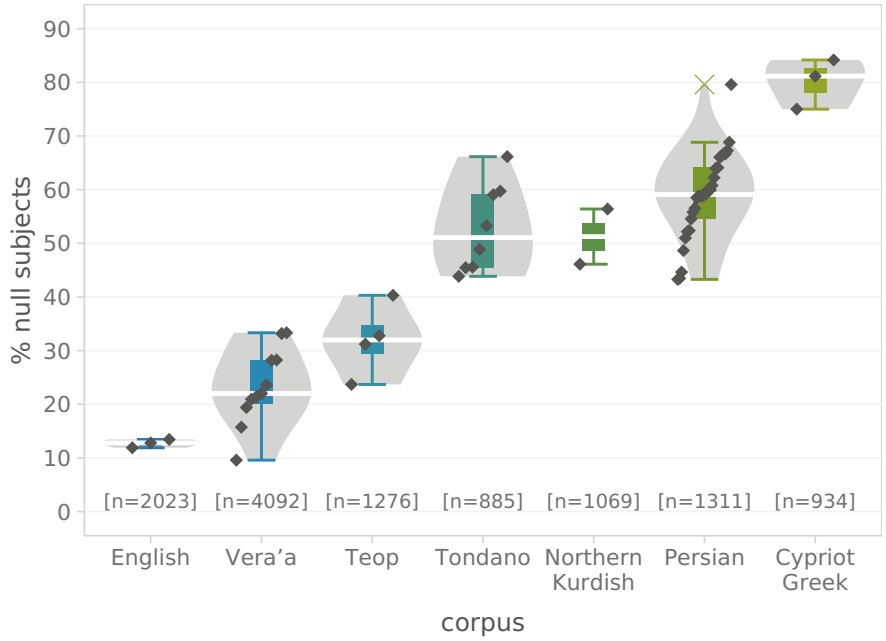

Figure 4 A more elaborate representation of Figure 3.

a subspecification of thereof in the `gfunc` column. It is wrapped in the function `grepl(pattern, data)`, which returns `TRUE` for every row that returns a match to its regular expression, and `FALSE` otherwise. In every row for which `grepl()` returns `TRUE`, that is, in every row that contains a GRAID subject gloss, we write `TRUE` to a new column `subj`. We can perform both the check and the assignment in a single step by using `:=`, *data.table's* referential assignment operator.

```
mc[grepl("^[as](_\\w*)?$", gfunc), subj := TRUE]
```

Let's look at a snippet from the data to confirm the procedure:

```
mc[corpus == "persian", ][1:12, c(1:3, 6:9, 12)]
```

```
     corpus    file uid   graid   gform ganim gfunc subj
 1: persian g1-f-01 001       #       #               NA
 2: persian g1-f-01 001 ln_deti ln_deti               NA
 3: persian g1-f-01 001  np.h:a      np     h     a TRUE
 4: persian g1-f-01 001     aux     aux               NA
 5: persian g1-f-01 001     adp     adp               NA
 6: persian g1-f-01 001  np:obl      np         obl   NA
 7: persian g1-f-01 001    np:p      np           p   NA
 8: persian g1-f-01 001  v:pred       v        pred   NA
 9: persian g1-f-01 002       #       #               NA
10: persian g1-f-01 002   other   other               NA
11: persian g1-f-01 002   0.h:s       0     h     s TRUE
12: persian g1-f-01 002     adp     adp               NA
```

Next we again make use of the `dt[i, j, by]` syntax with the `.N` shorthand for the number of rows in each group of the selection, as discussed in the previous example, to calculate the total number of subjects in the collection, split by corpus and file. As conditions we specify that all subjects should be selected (i.e. `subj` is `TRUE`), but excluding those that are clausal arguments (i.e. have ⟨`#`⟩ in the `gform` column).

```
all <- mc[subj == TRUE &
          !grepl("#", gform),
          .N,
          by = c("corpus", "file")]
```

We repeat the process with a different set of conditions to get the frequency of null subjects. Because matching ⟨`0`⟩ for zero and ⟨`#`⟩ for clause boundaries is mutually exclusive, we can drop the latter condition as we introduce the former.

```
zro <- mc[subj == TRUE &
          grepl("0", gform),
          .N,
          by = c("corpus", "file")]
```

The next step involves combining `zro` and `all` into a single table `null` via the `merge()` function, selecting both `corpus` and `file` as shared columns. We then rename the automatically generated columns containing the frequency data to `nNul` and `nAll`.

```
null <- merge(zro, all, by = c("corpus", "file"), all = TRUE)

setnames(null, c("N.x", "N.y"), c("nNul", "nAll"))
```

Now we can easily calculate the percentage of null subjects `pNul` by dividing the number of null subjects `nNul` by the number of all subjects `nAll` in each text:

```
null[, pNul := round(100 * nNul / nAll, 2)]

print(null)

      corpus      file nNul nAll  pNul
 1: cypgreek    jitros  198  244 81.15
 2: cypgreek    minaes  219  292 75.00
 3: cypgreek    psarin  335  398 84.17
---
58:    veraa      mvbw   81  343 23.62
59:    veraa     palaa   35  159 22.01
60:    veraa     palab   84  298 28.19
```

A summary of these data, split only by corpus, is given in Table 2. There is a striking spread of values, even for a sample of languages as small as those in Multi-CAST: from the infamous disinclination towards dropped subjects in English (12.7 %) to a remarkably strong preference for leaving subjects unexpressed in Cypriot Greek (80.5 %), and the remainder of corpora situated inbetween.

3.4 Visualization

For this graph, unlike Figure 1 above, we want to draw one boxplot per corpus, showing the distribution of null subjects across corpus texts. Additionally, we want to (A) order these boxplots from left to right in ascending order, sorted by the median percentage of null subject realization, and (B) show the

corpus	n(null)	n(all)	p(null)
C. Greek	752	934	80.5
English	256	2 023	12.7
N. Kurdish	542	1 069	50.7
Persian	778	1 311	59.3
Teop	375	1 276	29.4
Tondano	455	885	51.4
Vera'a	916	4 092	22.4

Table 2 Percentage of null subjects in the seven Multi-CAST corpora.

value for each corpus text as overlaid dots. For (A) we can again employ the `dt[i, j, by]` syntax that the *data.table* package provides, by this time specifying `median(pNul)` as the `j`:

```
mdn <- null[, median(pNul), by = "corpus"]

print(mdn)

     corpus    V1
1: cypgreek 81.15
2:  english 12.80
3:    nkurd 51.25
4:  persian 59.09
5:     teop 31.99
6:  tondano 51.07
7:    veraa 22.01
```

The table `mdn` has two columns: `corpus` and the automatically named `V1`, which contains the median values we just calculated. We sort the rows of `mdn` by the latter with the help of the `setorder()` function:

```
setorder(mdn, V1)

print(mdn)

     corpus    V1
1:  english 12.80
2:    veraa 22.01
3:     teop 31.99
4:  tondano 51.07
5:    nkurd 51.25
```

```
6:  persian 59.09
7: cypgreek 81.15
```

By default, the values of a vector or table column have no inherent hierarchy. We can assign an order to them by converting them to what R calls factors, and when doing so also specifying `ordered = TRUE` and an exhaustive, ordered list of values as `levels`. Here, we factorize the `corpus` column of table `null`, giving the sorted corpus labels in the first column of table `mdn` as factor levels:

```
null[, corpus := factor(corpus,
                        ordered = TRUE,
                        levels = mdn[, corpus])]
```

For (B), we do similarly with the `file` column, which contains the names of the various texts in each corpus. Here, we directly sort `null` first by corpus, then by the percentage of null subjects in each text, and then factorize accordingly:

```
setorder(null, corpus, pNul)

null[, file := factor(file,
                      ordered = TRUE,
                      levels = null[, file])]
```

With these preparations complete, it is time to load the *ggplot2* package, if this has not been done already:

```
library(ggplot2)
```

We then draw a series of boxplots from the data in `null`, with the values of `corpus` on the x-axis and the percentage of null subject realizations `pNul` on the y-axis. We further add an additional layer of point values with `geom_point()`. Because we factorized the values in the `corpus` and `file` columns, `ggplot()` will automatically draw them in the order we assigned to them. For `nul.plot` we also select suitable scale limits, choose axis titles, and apply a simple theme preset.

```
nul.plot <- ggplot(data = null, aes(x = corpus, y = pNul)) +
            geom_boxplot() +
            geom_point(aes(group = file),
                       position = position_dodge(width = 0.75),
                       shape = 1, size = 1.25) +
```

```
            scale_y_continuous(limits = c(0, 90),                    81
                               breaks = seq(0, 90, 20),              82
                               name = "% null subjects") +           83
            scale_x_discrete(name = "corpus") +                      84
            theme_bw()                                               85
                                                                     86
nul.plot                                                             87
```

The resulting plot `nul.plot` is reproduced in Figure 3, itself a less fancy rendition of Figure 4.

4 Case study: Information pressure

4.1 Theoretical background

One aspect of the investigation outlined in Section 2 above concerned how frequently transitive subjects (A) are realized lexically. As noted, the Multi-CAST data confirm the widely-known tendency for A to be generally non-lexical (i.e. pronominal or zero) as opposed to lexical, a tendency that is sometimes referred to as "avoid lexical A". With regards to explanations for this tendency, it has been proposed that the low levels of lexical realization of A is connected to information pressure, conceived as the density of lexical arguments contained within a single clause. In a transitive clause, containing an A and a P, it is assumed that the P will be generally lexical; see Table 1 and Figure 1 for confirmation of this. Therefore, one might assume that if a clause already contains a lexical P (for independent reasons), it is already 'saturated', and the A argument will be realized non-lexically to reduce information pressure within the clause. We can term this informally the "information pressure hypothesis" (see Haig & Schnell 2016b: 608 for details.)

In principle, there are four combinations of lexical and non-lexical A and P in transitive clauses, which are illustrated in (1):

(1) english_kent01_089 (Schiborr 2015; modified)

a. *the ferrets used to grab your hand, …*
```
## ln  np:a  lv  lv v:pred ln  np:p
```

b. *they used to grab your hand, …*
```
## pro:a lv  lv v:pred ln  np:p
```

c. *the ferrets used to grab it, …*
```
## ln  np:a  lv  lv v:pred pro:p
```

d. *they used to grab it, …*
```
## pro:a lv  lv v:pred pro:p
```

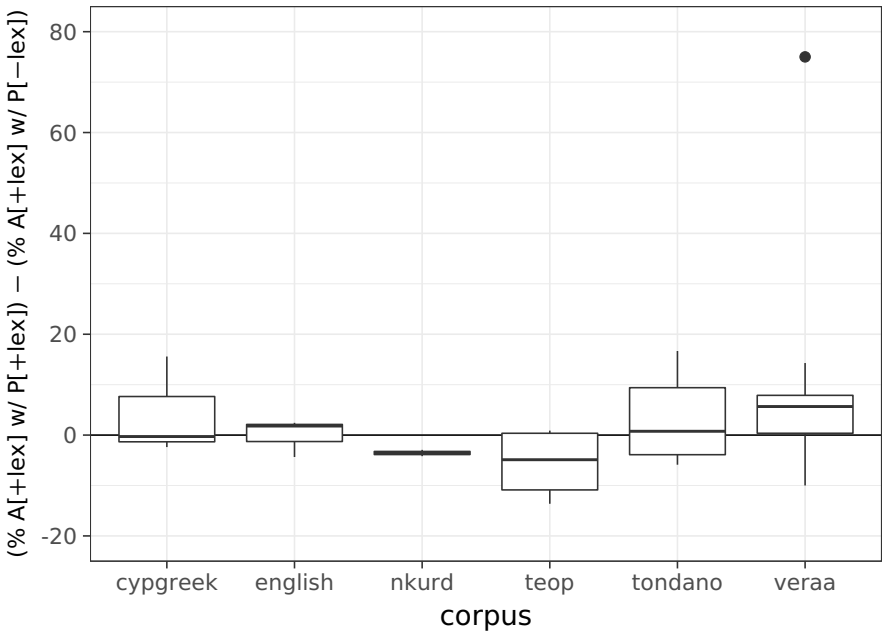

Figure 5 Distribution by corpus text of the percentage point differences between the lexicality of A in clauses with lexical and with non-lexical P, in six of the Multi-CAST corpora. Negative differences indicate a higher percentage of lexically expressed A in clauses with a non-lexical P argument, compared to clauses with lexical P arguments.

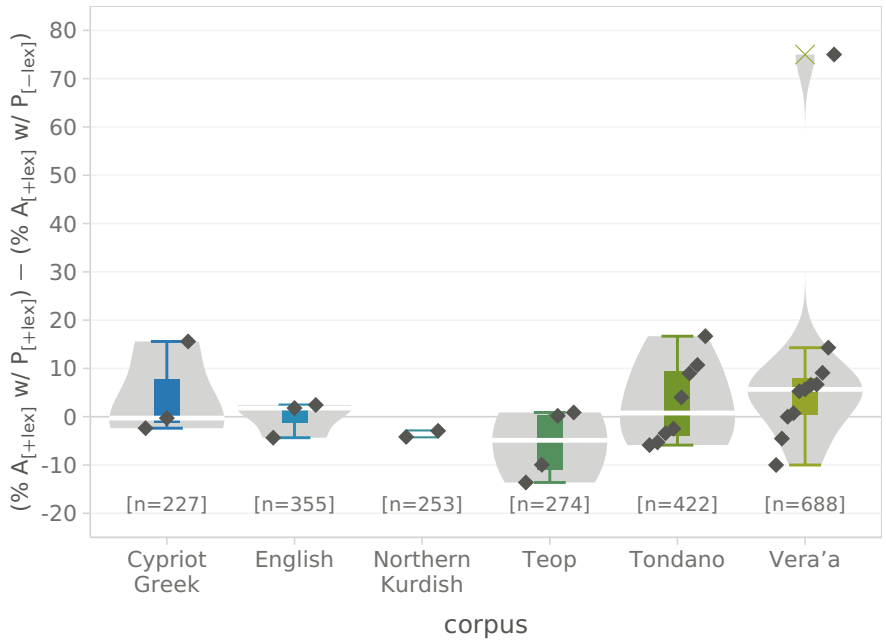

Figure 6 A more elaborate representation of Figure 5. Each text in the corpora is represented by a dot..

The information pressure hypothesis would predict that, all other things being equal, clauses which contain a lexical P (e.g. 1a and 1b above) would be less likely to also contain a lexical A, whereas clauses with a non-lexical P (1c) and (1d) would more readily accomodate a lexical A. Thus information pressure would predict that clauses such as (1a) would be statistically less frequent than clauses such as (1b). While no prediction would be made concerning (1c) and (1d), they provide a baseline against which general levels of lexical A can be compared.

This is a more complex case study than the previous two, for two reasons. First, we need to consider a clause and its constituents as a whole, rather than simply summing up individual constituent types within an entire corpus (e.g. the total number of P arguments in a corpus, etc.). Second, it raises a number of issues regarding the most appropriate statistical measures that can be applied. Here we restrict ourselves to a Fisher's Exact test as a measure of significance of correlation. The absolute figures are, as always, provided in the interest of future (re-)analysis. The steps involved in this investigation are spelled out in Sections 4.2 and 4.3, and visualized in Figures 5 and 6. The rationale and procedure behind the visualizations is described in Section 4.4.

The overall result is that with the exception of the Persian corpus (which is not included in Figures 5 and 5 for reasons explained below), the results of the significance tests do not allow us to assume with a sufficient degree of confidence that presence or absence of a lexical P in the same clause has any impact on the lexicality of the A. Both Haspelmath (2006) and Everett (2009) have reached a similar conclusions (cf. Haig & Schnell 2016b: 608 for a discussion), but our demonstration of it here is the most comprehensive and systematic investigation of the issue to date. Solely the results for the Persian corpus in Multi-CAST, however, are inconsistent with the overall findings; they can be attributed to the extremely short text lengths (little more than a minute in some cases) of the Persian texts, which leads to a preponderance of extreme values distorting the picture.[3] In summary, there is little indication that the lexicality of A is dependent on the lexicality of P in the same clause; we can thus dismiss information pressure as a motivator for the non-lexical A constraint.

In sum, the tendency to avoid lexical A is motivated by factors that are independent of whether the P argument is lexical or not. The strongest predictor that we have identified is the feature of humanness, which heavily correlates with the A role generally.

3 For an explanation of the outlier in the Vera'a corpus in Figure 6, see Section 4.4 below.

4.2 Association tests

In this example we will employ many of the same procedures as in the previous ones, but with an additional layer of complexity: we need to take into account elements of a clause other than the one we intend to count, that is, we need to look back and ahead within a clause. We have two goals: first, we attempt to test the associations implied by the information pressure hypothesis with the help of Fisher's exact test; second, we calculate the percentage point difference between the values, which we can then plot as a graph.

As before, we begin by bringing the *multicastR* package into the workspace, then retrieving the annotation data from the internet.

```
library(multicastR)

mc <- multicast("1606")
```

For each A argument in a clause, we need to identify the P argument in the same clause, if there is any. To do this, we first create a subset `qnt` of the annotation data `mc` that contains only rows with glosses that (i) have ⟨`:a`⟩ or ⟨`:p`⟩ function and (ii) are third person, or (iii) contain the clause boundary markers ⟨`##`⟩, ⟨`#`⟩, or ⟨`%`⟩.

For the condition targetting GRAID functions, we expand our regular expression to also match subspecifications of the basic ⟨`:a`⟩ and ⟨`:p`⟩ symbols: we match all those GRAID functions that start with the basic symbol, then optionally continue with a suffix delimited by an underscore ⟨`_`⟩. We also match secondary objects ⟨`:p2`⟩. The regular expressions for person and clause boundaries are similar to those discussed in the previous examples.

Crucially, we need to group the first two conditions in parentheses so that they are evaluated together, and separately from the third: we want to match rows that (have A or P function AND are third person), OR contain clause boundary markers.

```
qnt <- mc[(grepl("^[ap]2?(_\\w*)?$", gfunc) &
           grepl("^$|^h$|^d$", ganim)) |
           grepl("[#%]", gform), ]
```

The resulting subset looks as follows:

```
print(qnt[1:8, c(1, 4:9)])

     corpus word gloss    graid gform ganim gfunc
1: cypgreek    #     #       ##    ##
2: cypgreek    #     #       ##    ##
```

```
3: cypgreek    0  he_0    0.h:a     0     h     a
4: cypgreek  yon   son   np.h:p    np     h     p
5: cypgreek    #     #       ##    ##
6: cypgreek    0  he_0    0.h:a     0     h     a
7: cypgreek =ton  =him =pro.h:p  =pro     h     p
```

Next, we add a column to the `qnt` table with the help of *data.table's* referential assignment operator `:=`. This new column, `role`, we use to mark whether a row contains a gloss for an A or P argument, but we explicitly exclude clausal arguments such as complement clauses ⟨`#cc:p`⟩:

```
qnt[grepl("^a", gfunc) & !grepl("[#%]", gform), role := "A"]
qnt[grepl("^p", gfunc) & !grepl("[#%]", gform), role := "P"]
```

Then we add another column `lexA`, whose value we set to `TRUE` if a row has been marked as an A argument AND contains a lexical expression, this being one with a form gloss containing ⟨`np`⟩. If the former but not the latter condition is met, we instead set `lexA` to `FALSE`. For convenience, we can first simply set all rows with A arguments to `FALSE`, then selectively set the lexically expressed ones to `TRUE`:

```
qnt[role=="A", lexA := FALSE]
qnt[role=="A" & grepl("np", gform), lexA := TRUE]
```

We then repeat this procedure in column `lexP` for P arguments:

```
qnt[role=="P", lexP := FALSE]
qnt[role=="P" & grepl("np", gform), lexP := TRUE]
```

We now know which A and P arguments are within the scope of our analysis (i.e. not clausal), and which are lexical and which not. Before we proceed, let us take a look at `qnt`:

```
print(qnt[1:8, c(1, 4:9, 12:14)])

     corpus word gloss    graid gform ganim gfunc role  lexA  lexP
1: cypgreek    #     #       ##    ##                NA    NA    NA
2: cypgreek    #     #       ##    ##                NA    NA    NA
3: cypgreek    0  he_0    0.h:a     0     h     a    A FALSE    NA
4: cypgreek  yon   son   np.h:p    np     h     p    P    NA  TRUE
5: cypgreek    #     #       ##    ##                NA    NA    NA
6: cypgreek    0  he_0    0.h:a     0     h     a    A FALSE    NA
7: cypgreek =ton  =him =pro.h:p  =pro     h     p    P    NA FALSE
```

The next task involves establishing a relationship between those A and P arguments that are located in one and the same clause. In the previous steps, we have dropped all but those rows containing A and P arguments or clause boundary markers. We can now reasonably assume that if a row containing an A immediately follows or precedes a row containing a P, with no rows with clause boundaries in between, the two arguments must be elements of the same clause.

The `shift(x, n, type)` method provided by the *data.table* package allows us to evaluate the values in rows preceding and following a given row, that is, in figurative terms, "to look around". Here, `x` are our row selection criteria, `n` is the number of rows to shift, and `type` indicates the direction: `"lag"` looks behind, `"lead"` ahead.

In the following, for each row with an A argument, we mark in its `lexP` column (which until now contains only `NA`, see above) whether there is a non-lexical or lexical P argument in the the rows immediately above or below, which we check for using `shift()`. The conditions in the following two commands should be read as: if the grammatical role of the current row is A, AND (the preceding row contains a lexical P OR the following row contains a lexical P), then set `lexP` to `TRUE`, and vice versa for non-lexical P, for which we set `lexP` to `FALSE`. Should there be more than one P argument in the same clause, we understand the lexicality (versus non-lexicality) of any of them to be decisive, and hence we check for lexical P's second.

```
qnt[role=="A" &
    (shift(lexP, 1, type = "lag") == FALSE |
     shift(lexP, 1, type = "lead") == FALSE),
    lexP := FALSE]

qnt[role=="A" &
    (shift(lexP, 1, type = "lag") == TRUE |
     shift(lexP, 1, type = "lead") == TRUE),
    lexP := TRUE]
```

An excerpt from the result is given below. For each A argument in our sample, we now know whether it is lexically expressed or not (`lexA` is `TRUE` or `FALSE`), and whether it co-occurs with a lexical or non-lexical P in same clause (`lexP` is `TRUE` or `FALSE`), or not (`lexP` is `NA`).

```
print(qnt[1:8, c(1, 4:9, 12:14)])

     corpus word gloss    graid gform ganim gfunc role  lexA  lexP
1: cypgreek    #     #       ##    ##                NA    NA    NA
2: cypgreek    #     #       ##    ##                NA    NA    NA
3: cypgreek    0  he_0    0.h:a     0     h     a     A FALSE  TRUE
```

```
4: cypgreek  yon   son   np.h:p    np     h     p    P    NA  TRUE 48
5: cypgreek    #     #       ##    ##                  NA    NA    NA 49
6: cypgreek    0  he_0     0.h:a     0     h     a    A FALSE FALSE 50
7: cypgreek =ton  =him =pro.h:p  =pro     h     p    P    NA FALSE 51
```

It is important to note that this approach fails to capture A and P arguments that are separated by embedded clauses such as the one in (2):

(2) persian_g1-f-05_005 (Adibifar 2016)

bad	*yek*	*pesarbačeyi*			
then	one	little.boy			
## other	ln_deti	np.h:a			
ke	*bā*	*dočarxe*	*dāšt*	*rad*	*mišod*
that	with	bike	AUX.PST.3SG	passing	become
#rc ke	adp	np:obl	aux	lvc	v:pred %
češm	*=aš*	*in*	*sabadhā*	*=rā*	*gereft*
eye	=POSS.3SG	this	basket.PL	=ACC	catch.PST.3SG
lvc	=pro.h:poss	ln_dem	np:p	=rn_acc	v:pred

'A little boy who was passing by on a bike saw the baskets.'

While there are means and ways of working around this limitation, they are admittedly too complex to bring up here, and also prone to inaccuracies due to the inconsistent way GRAID marks the right-edge boundaries of consecutive clausal embeddings ⟨%⟩. While these drawbacks are unfortunate, they are unlikely to significantly affect our results, as complex clauses of the kind in (2) are likely to be uncommon.

As we are interested in which form A arguments take in a given environment, we next subset the data to include only rows for which both `lexA` and `lexP` are not `NA`:

```
qnt <- qnt[!is.na(lexA) & !is.na(lexP), ]                              52
                                                                       53
qnt[1:8, c(1, 4:9, 12:14)]                                             54
                                                                       55
     corpus word gloss graid gform ganim gfunc role  lexA  lexP       56
1: cypgreek    0  he_0 0.h:a     0     h     a    A FALSE  TRUE       57
2: cypgreek    0  he_0 0.h:a     0     h     a    A FALSE FALSE       58
3: cypgreek    0  he_0 0.h:a     0     h     a    A FALSE FALSE       59
4: cypgreek    0  he_0 0.h:a     0     h     a    A FALSE  TRUE       60
5: cypgreek    0 she_0 0.h:a     0     h     a    A FALSE FALSE       61
6: cypgreek    0 she_0 0.h:a     0     h     a    A FALSE  TRUE       62
7: cypgreek    0 she_0 0.h:a     0     h     a    A FALSE  TRUE       63
8: cypgreek    0 she_0 0.h:a     0     h     a    A FALSE  TRUE       64
```

Now that we have distilled our sample values from the raw data, we can begin with their analysis. The first step involves calculating the absolute frequencies of lexical and non-lexical A in clauses with lexical and non-lexical P in each corpus, for which we once again employ *data.table's* `dt[i, j, by]` syntax with `.N`, the shorthand for the number of rows in each `by` group of the selection:

```
fish <- qnt[, .N, by = c("corpus", "lexA", "lexP")]

print(fish)

      corpus  lexA  lexP   N
 1: cypgreek FALSE  TRUE 117
 2: cypgreek FALSE FALSE  77
 3: cypgreek  TRUE FALSE  15
 4: cypgreek  TRUE  TRUE  18
---
25:    veraa FALSE  TRUE 413
26:    veraa FALSE FALSE 245
27:    veraa  TRUE FALSE  15
28:    veraa  TRUE  TRUE  15
```

Table `fish` contains four rows for every Multi-CAST corpus, one for each combination of lexical and non-lexical A and P. Even at a glance, it is evident that lexical A is overall much less frequent than non-lexical A, as discussed above (i.e. the "avoid lexical A" constraint). We are interested in estimating the association between those values for which `lexP` is `TRUE` and `lexA` is `TRUE` or `FALSE`, and likewise for those where `lexP` is `FALSE`. To do this, we need the table in its wide form, with one row per corpus:

```
fish <- dcast(fish,
              corpus ~ lexA + lexP,
              value.var = "N")
```

The function `dcast(data, formula, value.var)` allows us to re-arrange the rows and columns in the table as per the `formula` and `value.var` given above. The left side of the formula defines the desired rows of the reshaped table, the right side its columns; note that what separates the two sides is a tilde `~`, not a minus sign. The argument `value.var` specifies the content of the table cells. Before we look at the re-shaped result, let's quickly rename the new column labels to something a little more concise:

```
setnames(fish, 2:5, c("-A-P", "-A+P", "+A-P", "+A+P"))

print(fish)

     corpus -A-P -A+P +A-P +A+P
1: cypgreek   77  117   15   18
2:  english  126  160   30   39
3:    nkurd   93  122   13   25
4:  persian  155  245   11   44
5:     teop  138  103   15   18
6:  tondano  193  201   15   13
7:    veraa  245  413   15   15
```

For our association tests we use Fisher's exact test, which is implemented in base R as `fisher.test()`. Given a contigency table of count data, Fisher's exact test estimates the independence of its rows and columns. `fisher.test()` accepts data in the form a matrix; because we are working with a `data.table`, we first need to convert our selection into a vector via `unlist()` (`data.table`s are structured internally like lists). We then turn this vector of length 4 into a 2×2 matrix via `matrix(x, ncol)`. Here we generate a matrix from the first row of table `fish`, containing the absolute frequencies for the Cypriot Greek corpus:

```
cyp <- matrix(unlist(fish[1, 2:5]), ncol = 2)

print(cyp)

     [,1] [,2]
[1,]   77   15
[2,]  117   18
```

We perform the test by passing this matrix to `fisher.test()`:

```
fisher.test(cyp)

Fishers Exact Test for Count Data

data:  cyp
p-value = 0.5683
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.3522746 1.7952621
sample estimates:
odds ratio
 0.7905871
```

The following `for` loop outputs the p-value for each of the Multi-CAST corpora in the sample:

```
for (i in 1:nrow(fish)) {
   tmp <- matrix(unlist(fish[i, 2:5]), ncol=2)

   print(paste(unlist(fish[i, 1]),
               "p =",
               round(fisher.test(tmp)$p.value, 5)))
}

[1] "cypgreek p = 0.5683"
[1] "english p = 1"
[1] "nkurd p = 0.37315"
[1] "persian p = 0.00689"
[1] "teop p = 0.26196"
[1] "tondano p = 0.6982"
[1] "veraa p = 0.17928"
```

As mentioned above, the results of the significance tests do not allow us to assume with a sufficient degree of confidence a difference in the lexicality of A for a given lexicality of P for any of the Multi-CAST corpora, perhaps with the exception of the Persian corpus, where short text lengths may be distorting the picture.

4.3 Percentage point differences

A slightly different approach is to compare directly the rates of lexical A in clauses with and without a lexical P, differentiating by corpus and individual texts. Recall that the information pressure hypothesis predicts that the percentage x of lexical A in clauses with a lexical P should be lower than the percentage y of lexical A in clauses without a lexical P. If we subtract the former from the latter (i.e. $x-y$), this ought therefore to yield a positive value. The results for analysis are provided in Figures 5 and 6 above, which illustrates that many of the values are negative, rather than predominantly positive. In this section we outline the steps necessary to conduct and visualize this analysis.

We have already observed above that the small sample size yielded by the Persian Pear film retellings fails to offer a nuanced picture. For this reason (and for the sake of simplicity), we begin by excluding the Persian data from the sample:

```
qnt <- qnt[corpus != "persian", ]
```

Using *data.table's* `dt[i, j, by]` syntax, we then count the number of A arguments in clauses with lexical and non-lexical P, split by corpus and file,

then repeat the process only for those A arguments that we have identified as lexical:

```
all <- qnt[, .N, by = c("corpus", "file", "lexP")]

lex <- qnt[lexA == TRUE,
           .N,
           by = c("corpus", "file", "lexP")]
```

We merge the two tables `all` and `lex` into a table `lxAP` by the `corpus`, `file`, and `lexP` columns, then rename the automatically generated column names to `nLexA` and `nAllA`:

```
lxAP <- merge(lex, all,
              by = c("corpus", "file", "lexP"),
              all = TRUE)

setnames(lxAP, c("N.x", "N.y"), c("nLexA", "nAllA"))
```

As the rate of lexical expression of the A role is expectably low ("avoid lexical A"), it is not unlikely for lexical A in some of the shorter texts in the MultiCAST collection to have a frequency of zero. In table `lxAP`, these texts will have `NA` in the `nLexA` column, which were introduced by specifying `all = TRUE` in `merge()` above. Let's have a look:

```
print(lxAP[is.na(nLexA), ])

     corpus    file  lexP nLexA nAllA
 1: tondano mapalus  TRUE    NA     6
 2: tondano   water FALSE    NA    16
 3:   veraa     anv  TRUE    NA    32
 4:   veraa     as1 FALSE    NA    15
 5:   veraa    gabg  TRUE    NA    12
 6:   veraa    gaqg  TRUE    NA    24
 7:   veraa    isam  TRUE    NA    18
 8:   veraa    mvbw FALSE    NA    24
 9:   veraa    mvbw  TRUE    NA    33
10:   veraa   palaa  TRUE    NA    16
```

A total of 10 of the 62 rows in table `lxAP` have missing values, many of them among the Vera'a texts. We replace these missing values with zero,

```
lxAP[is.na(nLexA), nLexA := 0]
```

so that we may then properly calculate the percentage of lexically expressed A in clauses with lexical and non-lexical P:

```
lxAP[, pLexA := round(100 * nLexA / nAllA, 2)]
```

The resulting table looks as follows.

```
print(lxAP)

      corpus      file  lexP nLexA nAllA pLexA
 1: cypgreek    jitros FALSE     2    21  9.52
 2: cypgreek    jitros  TRUE     5    42 11.90
 3: cypgreek    minaes FALSE     5    18 27.78
 4: cypgreek    minaes  TRUE     5    41 12.20
---
59:    veraa     palaa FALSE     1     7 14.29
60:    veraa     palaa  TRUE     0    16  0.00
61:    veraa     palab FALSE     1     9 11.11
62:    veraa     palab  TRUE     3    55  5.45
```

Those rows of the table that had `NA` previously now have an appropriate percentage value (i.e. 0 %); look, for instance, at the Vera'a text *palaa* in row 60. Had we not replaced the missing values, `pLexA` would display `NA`, as the division of `NA` by any number returns `NA`.

Next we re-shape the table with `dcast()` so that we have one row for each corpus file:

```
cast <- dcast(lxAP, corpus + file ~ lexP, value.var="pLexA")
```

The column names of the re-cast table are drawn from the values of the right side of the formula, in this case `lexP`. It is a good idea to give them less ambigious names with `setnames()`:

```
setnames(cast, c("FALSE", "TRUE"), c("lexP", "nonP"))
```

Finally, we calculate the percentage point difference by subtracting the percentage of lexical A in clauses with non-lexical P from the percentage of lexical A in clauses with lexical P:

```
cast[, ldif := lexP - nonP]
```

The hypothesis is that we are more likely to see a lower rate of lexical A if the P argument in the clause is lexical, as the notion of information pres-

sure places restrictions on the number of lexically expressed elements in a clause. By and large, we thus expect the values of `ldif` to be negative: negative differences indicate a higher percentage of lexically expressed A in clauses with a non-lexical P argument, compared to clauses with lexical P arguments.

```
print(cast)

      corpus      file  lexP  nonP   ldif
 1: cypgreek    jitros  9.52 11.90  -2.38
 2: cypgreek    minaes 27.78 12.20  15.58
 3: cypgreek    psarin 15.09 15.38  -0.29
 4:  english    kent01 17.39 21.74  -4.35
 5:  english   kent02a 17.65 15.22   2.43
---
27:    veraa      iswm  3.28  7.81  -4.53
28:    veraa       jjq  3.33  2.60   0.73
29:    veraa      mvbw  0.00  0.00   0.00
30:    veraa     palaa 14.29  0.00  14.29
31:    veraa     palab 11.11  5.45   5.66
```

Such does not seem to be the case, however: most of the values of `ldif` for the Multi-CAST texts hover around zero, with a median value of 0.73, a positive mean of 3.71, but also a fairly large standard deviation of 15.18 percentage points:

```
mean(cast[, ldif])
[1] 3.714194

sd(cast[, ldif])
[1] 15.17712

median(cast[, ldif])
[1] 0.73
```

These findings support the association tests we performed above: there is no strong evidence for a information pressure-based constraint on the lexical expression of A arguments. Table 3 summarizes these findings by corpus.

4.4 Visualization

As a final step, let's drive the point home by visualizing the distribution of percentage point differences across the Multi-CAST corpora, as done above in Figures 5 and 6. Once again we begin by loading the *ggplot2* package into the R workspace, if not done already:

	$P_{[+lex]}$			$P_{[-lex]}$			
corpus	$n(A_{[+lex]})$	n(all A)	$p(A_{[+lex]})$	$n(A_{[+lex]})$	n(all A)	$p(A_{[+lex]})$	diff.
C. Greek	18	135	13.33	15	92	16.30	+2.97
English	39	199	19.60	30	156	19.23	−0.37
N. Kurdish	25	147	17.01	13	106	12.26	−4.75
Teop	18	121	14.88	15	153	9.80	−5.08
Tondano	13	214	6.07	15	208	7.21	+1.14
Vera'a	15	428	3.50	15	260	5.77	+2.27

Table 3 Percentage of lexically expressed A in clauses with lexical and non-lexical P, and the difference between the two, in six of the Multi-CAST corpora.

```
library(ggplot2)
```

We draw a series of boxplots for each corpus in the sample, showing the distribution of the corpus texts. For completeness' sake, we add a horizontal line at `y = 0` with `geom_hline()`; this line should be drawn before the boxplots, or else it will overlap them.

```
qnt.plot <- ggplot(data = cast, aes(x = corpus, y = ldif)) +
            geom_hline(yintercept = 0) +
            geom_boxplot() +
            theme_bw()
```

We then adjust the scale limits, add axis titles, and slightly reduce the size of the y-axis title text:

```
qnt.plot <- qnt.plot +
            scale_y_continuous(
               limits=c(-20, 80),
               breaks=seq(-20, 80, 20),
               name="(% A[+lex] w/ P[+lex]) -
                     (% A[+lex] w/ P-[lex])"
            ) +
            scale_x_discrete(name = "corpus") +
            theme(axis.title.y = element_text(size = 9))

qnt.plot
```

The output of `qnt.plot` is reproduced as Figure 5, and Figure 6 above shows a fancier variant of the same plot that includes the full data as an overlay scatterplot.

With the exception of a single outlier in the Vera'a corpus, the distributions tightly cluster around zero, as expected. The outlier, the text *gabg*, has an unusually high percentage of lexical A in clauses with lexical P (75 %), but conversely has no occurrences of lexical A in clauses with non-lexical P. It is also one of the shorter texts in the Vera'a corpus, with a sample size of just four clauses with non-lexical P, three of whose A arguments happen to be lexical. This outlier is the reason for the suprisingly high standard deviation in `ldif` we noticed above.

Bibliography

The Multi-CAST collection

Haig, Geoffrey & Schnell, Stefan (eds.). 2015. *Multi-CAST: Multilingual Corpus of Annotated Spoken Texts*. (https://lac.uni-koeln.de/multicast/) (Accessed 2016-02-08).

Adibifar, Shirin. 2016. Multi-CAST Persian. In Haig, Geoffrey & Schnell, Stefan (eds.), *Multi-CAST: Multilingual corpus of annotated spoken texts*. (https://lac.uni-koeln.de/multicast-persian/) (Accessed 2016-07-01).

Brickell, Timothy C. 2016. Multi-CAST Tondano. In Haig, Geoffrey & Schnell, Stefan (eds.), *Multi-CAST: Multilingual corpus of annotated spoken texts*. (https://lac.uni-koeln.de/multicast-tondano/) (Accessed 2016-07-01).

Haig, Geoffrey & Thiele, Hanna. 2015. Multi-CAST Northern Kurdish. In Haig, Geoffrey & Schnell, Stefan (eds.), *Multi-CAST: Multilingual corpus of annotated spoken texts*. (https://lac.uni-koeln.de/multicast-northern-kurdish/) (Accessed 2016-02-22).

Mosel, Ulrike & Schnell, Stefan. 2015. Multi-CAST Teop. In Haig, Geoffrey & Schnell, Stefan (eds.), *Multi-CAST: Multilingual corpus of annotated spoken texts*. (https://lac.uni-koeln.de/multicast-teop/) (Accessed 2016-02-22).

Schiborr, Nils N. 2015. Multi-CAST English. In Haig, Geoffrey & Schnell, Stefan (eds.), *Multi-CAST: Multilingual corpus of annotated spoken texts*. (https://lac.uni-koeln.de/multicast-english/) (Accessed 2016-02-28).

Schnell, Stefan. 2015. Multi-CAST Vera'a. In Haig, Geoffrey & Schnell, Stefan (eds.), *Multi-CAST: Multilingual corpus of annotated spoken texts*. (https://lac.uni-koeln.de/multicast-veraa/) (Accessed 2016-02-22).

Vollmer, Maria C. & Hadjidas, Harris. 2015. Multi-CAST Cypriot Greek. In Haig, Geoffrey & Schnell, Stefan (eds.), *Multi-CAST: Multilingual corpus of annotated spoken texts*. (https://lac.uni-koeln.de/multicast-cypriot-greek/) (Accessed 2016-02-22).

Haig, Geoffrey & Schnell, Stefan. 2016a. Multi-CAST research context. In Haig, Geoffrey & Schnell, Stefan (eds.), *Multi-CAST: Multilingual corpus of annotated spoken texts*. (https://lac.uni-koeln.de/multicast/) (Accessed 2018-01-01).

Schiborr, Nils N. 2016. Multi-CAST structural overview. In Haig, Geoffrey & Schnell, Stefan (eds.), *Multi-CAST: Multilingual corpus of annotated spoken texts*. (https://lac.uni-koeln.de/multicast/).

Schiborr, Nils N. 2018. multicastR: A companion to the Multi-CAST collection. R package version 1.0.0. In Haig, Geoffrey & Schnell, Stefan (eds.), *Multi-CAST: Multilingual corpus of annotated spoken texts*. (https://cran.r-project.org/package=multicastR) (Accessed 2018-06-18).

References

Bickel, Balthasar. 2003. Referential density in discourse and syntactic typology. *Language* 79(4). 708–736.

Carvalho, Ana M. & Orozco, Rafael & Shin, Naomi L. (eds.). 2015. *Subject pronoun expression in Spanish: A cross-dialectal perspective*. Washington, D.C.: Georgetown University Press.

Chafe, Wallace. 1976. Givenness, contrastiveness, definiteness, subjects, topics, and point of view. In Li, Charles N. (ed.), *Subject and topic*, 25–55. New York: Academic Press.

Chafe, Wallace (ed.). 1980. *The Pear Stories: Cognitive, cultural, and linguistic aspects of narrative production*. Norwood, NJ: Ablex.

Chafe, Wallace. 1994. *Discourse, consciousness, and time: The flow and displacement of conscious experience in speaking and writing*. Chicago: The University of Chicago Press.

Dowle, Matt & Srinivasan, Arun. 2017. *data.table: Extension of 'data.frame'*. R package version 1.10.4. (`http://CRAN.R-project.org/package=data.table`) (Accessed 2018-01-21).

Du Bois, John. 1987. Absolutive zero: Paradigm adaptivity in Sacapultec Maya. *Lingua* 71(2). 203–222.

Du Bois, John. 2003. Argument structure: Grammar in use. In Du Bois, John & Kumpf, Lorraine & Ashby, William J. (eds.), *Preferred argument structure: Grammar as architecture for function*, 11–60. Amsterdam: John Benjamins.

Du Bois, John. 2017. Ergativity in discourse and grammar. In Coon, Jessica & Massam, Diane & Travis, Lisa D. (eds.), *The Oxford handbook of ergativity*, 23–57. Oxford: Oxford University Press.

Everett, Caleb. 2009. A reconsideration of the motivations for preferred argument structure. *Studies in Language* 33(1). 1–24.

Givón, Talmy. 1976. Topic, pronoun, and grammatical agreement. In Li, Charles N. (ed.), *Subject and topic*, 149–188. New York: Academic Press.

Givón, Talmy (ed.). 1979. *Discourse and syntax* (Syntax and semantics 12). New York: Academic Press.

Haig, Geoffrey & Schnell, Stefan. 2014. *Annotations using GRAID (Grammatical Relations and Animacy in Discourse): Introduction and guidelines for annotators (version 7.0)*. (`https://lac.uni-koeln.de/en/multicast/`) (Accessed 2015-12-30).

Haig, Geoffrey & Schnell, Stefan. 2016b. The discourse basis of ergativity revisited. *Language* 92(3). 591–618. (`https://doi.org/10.1353/lan.2016.0049`).

Haig, Geoffrey & Schnell, Stefan. 2016c. The discourse basis of ergativity revisited: Online appendices. *Language* 92(3). 1–14. (`https://doi.org/10.1353/lan.2016.0044`).

Haspelmath, Martin. 2006. Review of *Preferred argument structure: Grammar as architecture for function*, by John Du Bois, Lorraine Kumpf, and William Ashby. *Language* 82(4). 908–912.

Holmberg, Anders. 2009. Null subject parameters. In Biberauer, Theresa & Holmberg, Anders & Roberts, Ian & Sheehan, Michelle (eds.), *Parametric variation: Null subjects in minimalist theory*, 88–124. Cambridge: Cambridge University Press.

Lang, Duncan T. & CRAN team. 2016. *RCurl: General network (HTTP/FTP/...) client interface for R*. R package version 1.95-4.8. (`http://CRAN.R-project.org/package=RCurl`) (Accessed 2018-01-21).

Neeleman, Ad & Szendröi, Kriszta. 2008. Case morphology and radical pro-drop. In Biberauer, Theresa (ed.), *The limits of syntactic variation*, 331–348. Amsterdam: John Benjamins.

Perlmutter, David. 1971. *Deep and surface constraints in syntax*. New York: Holt, Rinehart and Winston.
Pešková, Andrea. 2013. Experimenting with pro-drop in Spanish. *SKY Journal of Linguistics* 26. 117–149.
Riester, Arndt & Baumann, Stefan. 2017. *The RefLex scheme — Annotation guidelines* (SinSpeC: Working papers of the SFB 732 14). Stuttgart: University of Stuttgart. (`http://elib.uni-stuttgart.de/handle/11682/9028`) (Accessed 2018-03-01).
Schiborr, Nils N. & Schnell, Stefan & Thiele, Hanna. 2018. *RefIND — Referent Indexing in Natural-language Discourse: Annotation guidelines (v1.1)*. Bamberg / Melbourne: University of Bamberg / University of Melbourne. (`https://www.uni-bamberg.de/fileadmin/aspra/misc/RefIND-guidelines-v1.1.pdf`) (Accessed 2018-04-09).
Stoll, Sabine & Bickel, Balthasar. 2009. How deep are differences in referential density? In Guo, Jiansheng & Lieven, Elena & Budwig, Nancy & Ervin-Tripp, Susan & Nakamura, Keiko & Özçaliskan, Seyda (eds.), *Crosslinguistic approaches to the psychology of language: Research in the tradition of dan isaac slobin*, 543–555. London: Psychology Press.
Wickham, Hadley. 2009. *ggplot2: Elegant graphics for data analysis*. New York: Springer. (`http://ggplot2.org`) (Accessed 2018-03-17).

Appendices

A List of GRAID symbols

The following is an inventory of the core GRAID symbols, reproduced from the *GRAID manual*, version 7.0 (Haig & Schnell 2014: 54–55). Language-specific additions are listed in the appendices of each corpus' *annotation notes*.

Form symbols

⟨0⟩	contrastively suppressed argument position ("zero")
⟨pro⟩	free pronoun in its full form
⟨pro-⟩	prefixed pronoun
⟨-pro⟩	suffixed pronoun
⟨pro=⟩	proclitic pronoun
⟨=pro⟩	enclitic pronoun
⟨np⟩	lexical NP
⟨refl⟩	overt reflexive or reciprocal pronoun
⟨w⟩	weak form, indicates a phonologically lighter form of a particular element which may under certain circumstances be realized as a clitic; attaches to other form glosses, e.g. ⟨wpro⟩
⟨v⟩	lexical verb as the form element of a predicate
⟨vother⟩	verbal element which may be used in predicative function, but lacks the normal means for assigning arguments, e.g. imperatives, participles, and certain types of nominalizations
⟨other⟩	form not relevant

Person-animacy symbols

⟨.1⟩	argument with first person reference
⟨.2⟩	argument with second person reference
⟨.h⟩	argument with human third person reference; non-human third person references are not glossed
⟨.d⟩	[optional] argument with anthropomorphized third person reference

Function symbols

⟨:a⟩	subject of a transitive clause
⟨:p⟩	object of a transitive clause
⟨:s⟩	subject of an intransitive clause
⟨:ncs⟩	non-canonical subject, i.e. an argument which lacks some or all of the morphological properties associated with subjects, but commands most of the syntactic properties associated with them in the language concerned
⟨:obl⟩	oblique argument, not goal or location
⟨:g⟩	oblique goal argument of a goal-oriented verb of motion, transitive or intransitive; also recipients and addressees
⟨:l⟩	oblique locative argument of a verb of location; also sources
⟨:poss⟩	possessor

⟨:appos⟩ apposition
⟨:dt⟩ dislocated topic
⟨:voc⟩ vocative, used for expressions denoting the individual to which an utterance is addressed
⟨:pred⟩ predicate of a clause
⟨:predex⟩ predicate of an existential expression
⟨:other⟩ function not relevant

Special form symbols

⟨aux⟩ auxiliary
⟨cop⟩ overt copular verb, usually in combination with a non-verbal predicate complement
⟨adp⟩ adposition
⟨ln⟩ NP-internal subconstituent occurring left of the phrasal head
⟨rn⟩ NP-internal subconstituent occurring right of the phrasal head
⟨lv⟩ subconstituent of a verb complex occurring left of the phrasal head
⟨rv⟩ subconstituent of a verb complex occurring right of the phrasal head
⟨other⟩ element not relevant; also a shorthand for ⟨other:other⟩
⟨nc⟩ element not considered / non-classifiable

Clause boundary markers and tags

⟨##⟩ left-edge boundary of a syntactically independent clause
⟨#⟩ left-edge boundary of all other clauses
⟨%⟩ right-edge boundary of an embedded clause, omitted if immediately followed by ⟨##⟩ or ⟨#⟩
⟨.neg⟩ negated clause
⟨ds⟩ clause containing direct speech; attaches to ⟨##⟩ and ⟨#⟩
⟨cc⟩ complement clause; attaches to ⟨#⟩
⟨ac⟩ adverbial clause; attaches to ⟨#⟩
⟨rc⟩ relative clause; attaches to ⟨#⟩
⟨nc⟩ clause not considered / non-classifiable; attaches to ⟨#⟩

B Regular expressions

The following list serves a quick reference to the syntax of regular expressions in ELAN and R. Bear in mind that this list is not comprehensive!

Note that in R, two backslashes instead of one are needed to escape characters, for instance `\\.` and `\\w` instead of `\.` and `\w`. Additionally, for certain functionality such as lookaround assertions to be usable in R, you need to switch to Perl-style regex by adding `perl=TRUE` as an argument to the functions `grepl()`, `sub()`, `regexpr()`, and so on. In the list below, the syntax for which this is needed is pointed out explicitly.

This short reference inherits much from Ulrike Mosel's highly recommendable guide *Searches with regular expressions in ELAN corpora*.[4]

Operators

`|`	separates alternatives, e.g. `yes|no` matches `yes` or `no`

Quantifiers

`?`	matches preceding text zero or one time, e.g. `a?` matches `a` or nothing
`*`	matches zero or more times
`+`	matches one or more times
`{n}`	matches exactly `n` times, e.g. `a{2}` only matches `aa`, but not `a`, `aaa`, …
`{x,}`	matches `x` or more times
`{,y}`	matches no more than `y` times
`{x,y}`	matches at least `x` times, but no more than `y` times
`?`	when added after other quantifiers, switches to minimal matching (vs. greedy, the default); e.g. where `ab*` matches as many repetitions of `ab` as it can before continuing, `ab*?` will only match the minimal number possible, i.e. once

Character classes

`[xyz]`	matches a single character out of `x`, `y`, and `z`
`[^xyz]`	matches everything but `x`, `y`, or `z`
`[a-z]`	hyphen indicates ranges, matches all lowercase characters
`[0-9]`	matches all digits
`[^A-Z]`	matches everything but uppercase characters
`[a-zA-Z]`	matches all lowercase and uppercase characters
`\w`	matches all alphanumeric characters and the underscore, shorthand for `[a-zA-Z0-9_]`
`\d`	matches all digits, shorthand for `[0-9]`
`\s`	matches all whitespace, shorthand for `[ \t\r\n\v\f]`
`\W`	matches everything but alphanumeric characters and the underscore, shorthand for `[^a-zA-Z0-9_]`

4 Available online at `https://tla.mpi.nl/wp-content/uploads/2011/12/Searches_in_ELAN_with_regular_expressions.pdf`.

`\D`	matches everything but digits, shorthand for `[^0-9]`
`\S`	matches everything but whitespace, shorthand for `[^ \t\r\n\v\f]`
`.`	matches any single character or whitespace

Groups and capturing

`(xyz)`	groups expressions and indexes them internally for backreferences
`\1`	backreference to the first group in in the expression; `\2` for the second, `\3` for the third, etc.
`(xyz)\1`	matches same pattern twice, i.e. `xyzxyz`
`(?:xyz)`	non-capturing group that does not create a backreference; in R with `perl=TRUE`

Boundaries and whitespace

`^`	matches the beginning of a line (with `\n` as delimiter) or cell
`$`	matches the end of a line (with `\n` as delimiter) or cell
`\A`	matches the beginning of a string or cell
`\z`	matches the end of a string or cell; in ELAN also equivalent to `\Z`
`\b`	matches zero-width word boundaries, i.e. the transitions between words and whitespace
`\t`	matches tab
`\r`	matches carriage return
`\n`	matches line feed; matches line breaks on Linux
`\r\n`	matches line breaks on Windows and Mac

Lookaround assertions

`x(?=y)`	zero-width positive lookahead, i.e. matches `x` if `y` follows, but does not 'use up' `y`; in R with `perl=TRUE`
`x(?!y)`	zero-width negative lookahead, i.e. matches `x` if no `y` follows, but does not 'use up' `y` if present; in R with `perl=TRUE`
`(?<=y)x`	zero-width positive lookbehind; in R with `perl=TRUE`
`(?<!y)x`	zero-width negative lookbehind; in R with `perl=TRUE`

Escape sequences

`\`	when placed before an active character (i.e. one with a special meaning in regular expressions), makes this character have its literal meaning; note that in R, escaping requires double backslashes
`\.`	matches a literal full stop; in R `\\.`
`\\`	matches a literal backslash; in R `\\\`

Multi-CAST

Multilingual Corpus of Annotated Spoken Texts

lac.uni-koeln.de/multicast/